JU_Jabian_6.1 Code Template

Table of Contents

Data Structures

Template with PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;template <class T>
using ordered_set = tree<T,null_type,less<T>, rb_tree_tag, tree_order_statistics_node_update>;
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'  #define F first  #define S second
#define all(s) (s).begin(), (s).end()
#define rall(s) s.rbegin(), s.rend()
#define sz(x) (int)x.size()
#define mem(a, b) memset(a, b, sizeof(a))
#define cnt_bit(x) __builtin_popcountll(x)
#define getbit(n, i) (((n) & (1LL << (i))) != 0)
#define msb(x) 31 - __builtin_clz(x)
typedef array<int,3> tu;
mt19937_64 gen(random_device{}());int r_num = uniform_int_distribution<long long>{0, 1e18}(gen);
view->layout 3 column then group max 2 column
Sublime text setup ->tool->build system->newbuild
{
    "shell_cmd": "g++ \"${file}\" -o a.exe && a.exe < input.in > output.in"
}
{// run with    ..
"cmd": ["g++.exe","-std=c++14", "${file}", "-o", "${file_base_name}.exe", "&&" , "${file_base_name}.exe<input.in
    output.in"],
"selector":"source.cpp",
"shell":true,
"working_dir":"$file_path",
"file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$"
}
```

2D BIT

```
template <typename T> class BIT2D {
  private:
  const int n, m;
  vector<vector<T>> bit;
  public:
  BIT2D(int n, int m) : n(n), m(m), bit(n + 1, vector<T>(m + 1)) {}
  void add(int r, int c, T val) {
    r++, c++;
    for (; r <= n; r += r & -r) {
      for (int i = c; i <= m; i += i & -i) { bit[r][i] += val; }
    }
  }
  T rect_sum(int r, int c) {
    r++, c++;
    T sum = 0;
    for (; r > 0; r -= r & -r) {
      for (int i = c; i > 0; i -= i & -i) { sum += bit[r][i]; }
    }
    return sum;
  }
  T rect_sum(int r1, int c1, int r2, int c2) {
    return rect_sum(r2,c2)-rect_sum(r2,c1-1)-rect_sum(r1-1,c2)+rect_sum(r1-1,c1-1);
  }
};
vector<vector<int>> ar(n,vector<int>(m)); BIT2D<int> bit(n, m); bit.add,bit.rect_sum
```

BIT

```
ll tree[N+5];
void update(int pos, ll val){
    while (pos <= N)// must N = 4 * n
    {
        tree[pos] += val;
        pos += (pos & -pos);
    }
}
ll query(int pos){
    ll sum = 0;
    while (pos > 0)
    {
        sum += tree[pos];
        pos -= (pos & -pos);
    }
    return sum;
}
ll sum(int l,int r){
  return query(r) - query(l-1);
}
```

Sparse Table

```
int spt[N][22];
int n,ar[N];
int combine(int a,int b){
  return min(a , b);
}
void buildST(){
  for (int i = 0; i < n; i++) spt[i][0] = ar[i];

  for (int j = 1; (1 << j) <= n; j++) {
    for (int i = 0; (i + (1 << j) - 1) < n; i++) {
            spt[i][j] = combine(spt[i + (1 << (j - 1))][j - 1] , spt[i][j - 1]);
    }
  }
}
int query(int l, int r){
```

```
    if(l>r) return INT_MAX;
  int j = (int)log2(r - l + 1);
  return combine(spt[l][j], spt[r - (1 << j) + 1][j]);
}
//! 0 based index -> query O(1),pre compute nlog(n) and support idempotent function f(a,a) = a
//! minimum, maximum, GCD, and XOR buildST();
```

Segment Tree

```
int ar[N],tree[4*N],lazy[4*N];// lazy seg start
int combine(int x,int y){return min(x , y);}
void build(int node,int l,int r){
    lazy[node] = -1;//! change
    if(l == r){tree[node] = ar[l];return;}
    int mid = (l + r)/2;
    build(node*2,l,mid);build(node*2+1,mid+1,r);
    tree[node] = combine(tree[node*2],tree[node*2 + 1]);
}
void propagate(int node,int l,int r){
    if(lazy[node] == -1)return;//! change
    if(l != r){lazy[2*node] = lazy[node];//! change
        lazy[2*node+1] = lazy[node];}
    tree[node] = lazy[node];//! change
    lazy[node] = -1; //!change
}
int query(int node,int l,int r,int a,int b){
    propagate(node,l,r);
    if(l > b || r < a){ return  oo;}
    if(a <= l && r <= b){return tree[node];}
    int mid = (l + r)/2;
    int x = query(node * 2,l,mid,a,b);
    int y = query(node * 2 + 1,mid + 1,r,a,b);
    return combine(x,y);
}
void update(int node,int l,int r,int a,int b,int val){
    propagate(node,l,r); if(a > r || b < l)return;
    if(a <= l && r <= b){
        lazy[node] = val;//! change
        propagate(node,l,r); return;
    }
    int mid = (l + r)/2; update(node*2,l,mid,a,b,val);
    update(node*2+1,mid+1,r,a,b,val);
    tree[node] = combine(tree[node*2],tree[node*2+1]);
}
ll tree[M],ar[M];// just seg Tree start
ll combine(int a,int b){return max(a,b);}
void init(int node,int b,int e){
    if(b == e){tree[node] = ar[b];return;}
    int Left = node * 2;int Right = node * 2 + 1;
    int mid = (b+e)/2; init(Left,b,mid); init(Right,mid+1,e);
    tree[node] = combine(tree[Left] , tree[Right]);
}
ll query(int node,int b,int e,int i,int j){
    if(i>e || j<b)return 0;if(i<=b && j>=e){
        return tree[node];
    }
    int Left = node * 2;int Right = node * 2 + 1;
    int mid = (b+e)/2;ll p1 = query(Left,b,mid,i,j);
    ll p2 = query(Right,mid+1,e,i,j);return combine(p1,p2);
}
void upgrade(int node,int b,int e,int i,int newValue){
    if(b == e){tree[node] = newValue;ar[b] = newValue;
    return;
    }
    int Left = node * 2;int Right = node * 2 + 1;int mid = (b+e)/2;
    if(i <= mid)upgrade(Left,b,mid,i,newValue);
    else upgrade(Right,mid+1,e,i,newValue);
    tree[node] = combine(tree[Left] , tree[Right]);
}
// Iterative Segment Tree ->
int combine(int a,int b){return max(a,b);}
void build(int n){
    for(int i=1;i<=n;i++) tree[n+i-1]=ar[i];
    for(int i=n-1;i;i--) tree[i]=combine(tree[i<<1],tree[i<<1|1]);
}
void update(int p,int v,int n){
    for(tree[p+=n-1]=v,p>>=1;p;p>>=1){
        tree[p]=combine(tree[p<<1],tree[p<<1|1]);
    }
}
int query(int l,int r,int n){
    int ans=NEUTRAL;
    for(l+=n-1,r+=n-1;l<=r;l>>=1,r>>=1){
        if(l&1) ans=combine(ans,tree[l++]);
        if(!(r&1)) ans=combine(ans,tree[r--]);
    }return ans;
}
```

Persistent Segment Tree

```
struct node{
    int val;node* lc;node* rc;
};
int ar[N];
node* versions[N+5];
vector<node*> lagbe;
node* build(int l,int r){
    if(l == r){
        node* me = new node();lagbe.pb(me);
        me->val = ar[l];  me->lc = NULL;
        me->rc = NULL;return me;
    }
    node* my = new node();lagbe.pb(my);
    int mid = (l + r)/2;
```

```
    my->lc = build(l,mid);
    my->rc = build(mid+1,r);
    my->val = (my->lc->val + my->rc->val); return my;}
int query(node* my,int l,int r,int L,int R){
    if(R < l || r < L)return 0;
    if(L <= l && r <= R){return my->val;}
    int mid = (l + r)/2;
    int p = query(my->lc,l,mid,L,R);
    int q = query(my->rc,mid+1,r,L,R);
    return p + q;}
node* update(node* my,int l,int r,int ind,int val){
    if(l == r){
        node* new_my = new node(); lagbe.pb(new_my);
        new_my->lc = NULL;new_my->rc = NULL;new_my->val = val;
        return new_my; }
    node* new_me = new node();
    lagbe.pb(new_me);
    int mid = (l + r)/2;
    if(ind <= mid){
        node* x = update(my->lc,l,mid,ind,val);
        new_me->lc = x;
        new_me->rc = my->rc;}
    else{
        node* y = update(my->rc,mid+1,r,ind,val);
        new_me->lc = my->lc;
        new_me->rc = y;
    }
    new_me->val = new_me->lc->val + new_me->rc->val;
    return new_me;
}
// after test case delete all node* ->for(auto u : lagbe){delete u; }lagbe.clear();
int now = 1;versions[now] = build(1,n);versions[k] = update(versions[k],1,n,a,x);
```

Merge Sort Tree

```
struct Marge_Sort_Tree {
    int n;
    vector<vector<int>> tree;
    void build(vector<int> &a, int x, int l, int r) {
        if (l + 1 == r) {
            tree[x] = {a[l]};
            return;
        }

        int m = (l + r) / 2;
        build(a, 2 * x + 1, l, m);
        build(a, 2 * x + 2, m, r);
        merge(all(tree[2 * x + 1]), all(tree[2 * x + 2]), back_inserter(tree[x]));
    }
    Marge_Sort_Tree(vector<int>& a) : n(a.size()) {
        int SIZE = 1 << (__lg(n) + bool(__builtin_popcount(n) - 1));
        tree.resize(2 * SIZE - 1);
        build(a, 0, 0, n);
    }
    int count(int lq, int rq, int mn, int mx, int x, int l, int r) {
        if (rq <= l || r <= lq) return 0;
        if (lq <= l && r <= rq) return lower_bound(all(tree[x]), mx) - lower_bound(all(tree[x]), mn);

        int m = (l + r) / 2;
        int a = count(lq, rq, mn, mx, 2 * x + 1, l, m);
        int b = count(lq, rq, mn, mx, 2 * x + 2, m, r);
        return a + b;
    }
    int count(int lq, int rq, int mn, int mx) {
        return count(lq - 1, rq, mn, mx + 1, 0, 0, n);// query 1 base call l to r how many mn to mx number
    }
};
```

Trie

```
struct Node {
  Node* links[26];int cnt_pre = 0;int cnt_word = 0;
  bool contains(char c) { return (links[c-'a'] != NULL);}
  void put(char c, Node* node) {links[c - 'a'] = node;}
  Node* get(char c) { return links[c - 'a']; }
  void increasePre() { cnt_pre++; }
  void increaseWord() { cnt_word++; }
  void decreasePre() { cnt_pre--; }
  void decreaseWord() { cnt_word--; }
  int getPre() { return cnt_pre; }
  int getWord() { return cnt_word; }
  void deleteNode(){
        for(int i = 0; i < 26; ++i) {
            if(links[i] != nullptr) { links[i]->deleteNode(); delete links[i];}
        }
    }
};
class Trie {
private:
  Node *root;
public:
  Trie() {
    root = new Node();
  }
  void deleteNode(){ root->deleteNode();delete root; }
  void insert(string s) {
    Node* cur_node = root;
    for(char c : s) {
      if(!cur_node -> contains(c)) {
        cur_node -> put(c, new Node());
      }//! move reference node
      cur_node = cur_node -> get(c);
      cur_node -> increasePre();
    }
```

```
    cur_node -> increaseWord();
  }
  int countWords(string s) {
    Node* cur_node = root;
    for(char c : s) {
      if(!cur_node -> contains(c)) {
        return 0;
      }
      cur_node = cur_node -> get(c);
    }
    return cur_node -> getWord();
  }
  int countPrefixes(string s) {
    Node* cur_node = root;
    for(char c : s) {
      if(!cur_node -> contains(c)) {
        return 0;
      }
      cur_node = cur_node -> get(c);
    }
    return cur_node -> getPre();
  }
  void erase(string s) {
    Node* cur_node = root;
    for(char c : s) {
      if(!cur_node -> contains(c)) {
        return;
      }
      cur_node = cur_node -> get(c);
      cur_node -> decreasePre();
    }
    cur_node -> decreaseWord();
  }
  int getMax(int num) {
    Node* cur_node = root;
    int mx = 0;
    for(int i = 31; i >= 0; i--) {
      int bit = (num >> i) & 1;
      if(cur_node -> contains(bit ^ 1) && cur_node -> getCnt(bit ^ 1) > 0) {
        mx = ((1LL << i) | mx);
        cur_node = cur_node -> get(bit ^ 1);
      } else {
        cur_node = cur_node -> get(bit);
      }
    }
    return mx;
  }
};
```

Graph Algorithms

DSU

```
class DSU{
    vector<int> parent,Size;
public:
    DSU(int n){
        for(int i = 0; i <= n; i++){
            parent.push_back(i);
            Size.push_back(1);
        }
    }
    int find_par(int node){
        if(parent[node] == node){
            return node;
        }
        return parent[node] = find_par(parent[node]);
    }
    void union_size(int u, int v) {
        int pu = find_par(u);
        int pv = find_par(v);
        if(pu == pv)return;

        if(Size[pu] < Size[pv]){
            parent[pu] = pv;
            Size[pv] += Size[pu];
        }
        else{
            parent[pv] = pu;
            Size[pu] += Size[pv];
        }
    }
    int size(int node){
        return Size[find_par(node)];}
};
```

Dijkstra

```
void dijkstra(int s,int n,vi &dis,vector<vector<pii>> &graph){
    priority_queue<pii,vector<pii>,greater<pii>> pq;
    for(int i = 1; i <= n; i++){
        dis[i]=oo;
    }
    dis[s]=0;
    pq.push({0,s});
    while(!pq.empty()){
        int a = pq.top().S,co = pq.top().F;
        pq.pop();
        if(co > dis[a]) continue;
        for (auto u: graph[a]){
            int b = u.F,w = u.S;
            if (dis[a]+w < dis[b]){
                dis[b] = dis[a]+w;
```

```
                pq.push({dis[b],b});
            }
        }
    }
}
```

Bellman Ford

```
struct edge {
  int u,v,w,x;
};
vector<edge> e;
int dist[N],par[N],neg[N];
void BellmenFord(int s,int n,int m) {
  for(int i = 0; i <= n; i++) {
    dist[i] = oo,neg[i] = 0;
  }
  dist[s] = 0;
  for(int i = 1; i <= n; i++) {
    for(int j = 0; j < m; j++) {
      int u = e[j].u,v = e[j].v,w = e[j].w;
      if(dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        par[v] = u;
        if(i == n) {
          neg[v] = 1;
        }
      }
    }
  }
}// neg[i] tells it is a part of negative cycle or not
```

Floyd Warshall

```
for(int i = 1; i <= n; i++) {
  for(int j = 1; j <= n; j++) {
    nxt[i][j] = j;
  }
}
for(int k = 1; k <= n; k++) {
  for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
      if(dist[i][k] + dist[k][j] < dist[i][j]) {
        dist[i][j] = dist[i][k] + dist[k][j];
        nxt[i][j] = nxt[i][k];
      }
    }
  }
}
int s = 1, t = 4;
cout << s << " ";
int cur = s;
while(cur != t) {
  cur = nxt[cur][t];
  cout << cur << " ";
}
```

Articulation Point

```
vi graph[N];set<int> point;
int low[N],tin[N],vis[N],timer;
void dfs(int node,int par){
  vis[node] = 1;
  low[node] = tin[node] = ++timer;
  int child = 0;
  for(auto u : graph[node]) {
    if(u == par)continue;
    if(vis[u])low[node] = min(low[node],tin[u]);
    else {
      child++;
      dfs(u,node);
      low[node] = min(low[node],low[u]);
      if(low[u] >= tin[node] && par != -1)point.insert(node);
    }
  }
  if(child > 1 && par == -1)point.insert(node);
}
```

Bridge

```
set<pii> bridge;
int low[N],tin[N],vis[N],timer = 0;
void dfs(int node,int par){
  vis[node] = 1;
  low[node] = tin[node] = ++timer;
  for(auto child : graph[node]) {
    if(child == par)continue;
    if(vis[child]) {
      low[node] = min(low[node], tin[child]);
    }
    else {
      dfs(child,node);
      low[node] = min(low[node], low[child]);
      if(low[child] > tin[node]) {
        int x = min(node,child),y = max(node,child);
        bridge.insert({x,y});
      }
    }
  }
}
```

SCC

```cpp
// given a directed graph return the minimum number of edges to be added so that the whole graph  become an SCC
bool vis[N];
vector<int> g[N], r[N], G[N], vec; //g is the condensed graph
void dfs1(int u) {
  vis[u] = 1;
  for(auto v: g[u]) if(!vis[v]) dfs1(v);
  vec.push_back(u);
}
vector<int> comp;
void dfs2(int u) {
  comp.push_back(u);
  vis[u] = 1;
  for(auto v: r[u]) if(!vis[v]) dfs2(v);
}
int idx[N], in[N], out[N];
int main() {
   int n, m;
  cin >> n >> m;
  for(int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    r[v].push_back(u);
  }
  for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
  reverse(vec.begin(), vec.end());
  memset(vis, 0, sizeof vis);
  int scc = 0;
  for(auto u: vec) {
    if(!vis[u]) {
      comp.clear();
      dfs2(u);
      scc++;
      for(auto x: comp) idx[x]=scc;
    }
  }
  for(int u = 1; u <= n; u++) {
    for(auto v: g[u]) {
      if(idx[u] != idx[v]) {
        in[idx[v]]++, out[idx[u]]++;
        G[idx[u]].push_back(idx[v]);
      }
    }
  }
  int needed_in=0, needed_out=0;
  for(int i = 1; i <= scc; i++) {
    if(!in[i]) needed_in++;
    if(!out[i]) needed_out++;
  }
  int ans = max(needed_in, needed_out);
  if(scc == 1) ans = 0;
  cout << ans << '\n';
}
```

BCC

```cpp
/bcc[i] is which num Biconnected element it
set<int> g[N];//use set avoid multiple same edges
vector<int> bcc[N], st;
int low[N], dis[N], T = 0, sz = 0;
vector<vector<int>> allBCC;
void dfs(int u, int pre = -1) {
    low[u] = dis[u] = ++T;
    st.push_back(u);
    for(auto v : g[u]) {
        if(!dis[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dis[u]) {
                sz++;vector<int> comp;
                int x;
                do{
                    x = st.back();
                    st.pop_back();
                    bcc[x].push_back(sz);
                    comp.push_back(x);
                } while (x != v);
                bcc[u].push_back(sz);
                comp.push_back(u);
                sort(comp.begin(), comp.end());
                comp.erase(unique(comp.begin(), comp.end()), comp.end());
                allBCC.push_back(comp);
            }
        }
        else if(v != pre)
            low[u] = min(low[u], dis[v]);
    }
}
void Goriber_solve(){
    int n,m;cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;cin >> u >> v;
        if (u == v) continue;
        g[u].insert(v);g[v].insert(u);
    }
    for(int i = 0; i < n; i++)if(!dis[i])dfs(i);
    for (int i = 0; i < n; i++) {
        if(g[i].empty()) {sz++;
            allBCC.push_back({i});
        }
    }
    cout << allBCC.size() << "\n";
    for(auto &comp : allBCC) {
```

```
        cout << comp.size();
        for(int x : comp) cout << " " << x;
        cout << "\n";
    }
}//if bcc[u].size() > 1 it is an AP
```

Topological Sort

```
void dfs(int node) {
  vis[node] = 1;
  for(auto u : graph[node]) {
    if(!vis[u]){ dfs(u); }
  } ans.pb(node);
}
void Goriber_solve() {
  int n,m;cin >> n >> m;bool ok = 0;
  for(int i = 0; i < m; i++) {
    int u,v;cin >> u >> v;
    if(u == v)ok = 1;graph[u].pb(v);
  }
  if(ok) {cout << "IMPOSSIBLE\n";return;}
  for(int i = 1; i <= n; i++) {
    if(!vis[i]) {dfs(i);}
  }
  reverse(all(ans));vi pos(n+1);
  for(int i = 0; i < sz(ans); i++) {
    pos[ans[i]] = i+1;
  }
  for(int i = 1; i <= n; i++) {
    for(auto u : graph[i]) {
      if(pos[i] > pos[u]) {cout << "IMPOSSIBLE\n";return;
      }
    }
  }
  for(auto u : ans) {cout << u << " ";}cout << endl;
}
```

Cycle Detection

```
vi graph[N];// for directed graph
int vis[N],start = -1,finish,par[N];
void dfs(int node){
    vis[node] = 1;
    for(auto child : graph[node]){
        if(vis[child] == 0){
            par[child] = node;
            dfs(child);
            if(start != -1)return;
        }
        else if(vis[child] == 1){
            start = child;
            finish = node;
            return;
        }
    }
    vis[node] = 2;
}
bool is_cycle(int node,int par) {// for undirtected graph
  vis[node] = true;
  ans.pb(node);
  for(int child : graph[node]) {
    if(child == par) continue;
    if(vis[child] == 0) {
      if(is_cycle(child,node) == 1)return 1;
    }
    else {
      ans.pb(child);
      return 1;
    }
  }
  ans.pop_back();
  return 0;
}
//main function->
for(int i = 1; i <= n; i++){
  if(vis[i] == 0){{dfs(i);if(start != -1)break;}
int x = ans.back();vi path;path.pb(x);ans.pop_back();int node = ans.back();
while(node != x) {
  path.pb(node);ans.pop_back();node = ans.back();
}path.pb(x);
```

LCA

```
vector<int> graph[N];
int Level[N];int dp[N][22];
void dfs(int node,int par,int lev){
    dp[node][0] = par;
    Level[node] = lev;
    for(int i = 1; i <= 20; i++){
        dp[node][i] = dp[dp[node][i-1]][i-1];
    }
    for(auto child : graph[node]){
        if(child != par){
            dfs(child,node,lev+1);
        }
    }
}
int get_kth_par(int node,int k){
    for(int i = 20; i >= 0; i--){
        if((k >> i) & 1){
            node = dp[node][i];
        }
    }return node;
```

```
}
int get_lca(int a,int b){
    if(Level[a] > Level[b]){
        swap(a,b);
    }
    int k = Level[b] - Level[a];
    b = get_kth_par(b,k);
    if(a == b){ return a;}
    for(int i = 20; i >= 0; i--){
        if(dp[a][i] != dp[b][i]){
            a = dp[a][i];
            b = dp[b][i];
        }
    } return dp[a][0];
}  //dfs(1,0,0);
```

Bipartite Matching

```
const int N = 3e5 + 9;// Time complexity O(E*sqrt(V))
struct HopcroftKarp {
  static const int inf = 1e9;int n;
  vector<int> l, r, d;vector<vector<int>> g;
  HopcroftKarp(int _n, int _m) {
    n = _n;int p = _n + _m + 1;g.resize(p);
    l.resize(p, 0);r.resize(p, 0);d.resize(p, 0);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v + n); //right id is increased by n, so is l[u]
  }
  bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
      if (!l[u]){d[u] = 0; q.push(u);}
      else d[u] = inf;
    } d[0] = inf;
    while (!q.empty()) {
      int u = q.front();q.pop();
      for (auto v : g[u]) {
        if (d[r[v]] == inf) {
          d[r[v]] = d[u] + 1;q.push(r[v]);
        }
      }
    }
    return d[0] != inf;
  }
  bool dfs(int u) {
    if (!u) return true;
    for (auto v : g[u]) {
      if(d[r[v]] == d[u] + 1 && dfs(r[v])) {
        l[u] = v;r[v] = u;return true;
      }
    } d[u] = inf;
    return false;
  }
  int maximum_matching() {
    int ans = 0;
    while (bfs()) {
      for(int u = 1; u <= n; u++)if(!l[u] && dfs(u)) { ans++; }
    } return ans;
  }
  vector<pii> get_pairs() {
    vector<pii> pairs;
    for(int u = 1; u <= n; u++) {
      if(l[u]) {
        pairs.emplace_back(u,l[u] - n);
      }
    } return pairs;
  }
};
```

Max Flow (Dinic)

```
const int N = 5010; const long long inf = 1LL << 61;
struct Dinic {
  struct edge {
    int to, rev;long long flow, w;int id;
  };
  int n, s, t, mxid;vector<int> d, flow_through;
  vector<int> done; vector<vector<edge>> g;
  Dinic() {}
  Dinic(int _n) {
    n = _n + 10;mxid = 0;g.resize(n);
  }
  void add_edge(int u, int v, long long w, int id = -1) {
    edge a = {v, (int)g[v].size(), 0, w, id};
    edge b = {u, (int)g[u].size(), 0, 0, -2};//for bidirectional edges cap(b) = w
    g[u].emplace_back(a);g[v].emplace_back(b);mxid = max(mxid, id);
  }
  bool bfs() {
    d.assign(n, -1);d[s] = 0;queue<int> q;q.push(s);
    while (!q.empty()) {
      int u = q.front();q.pop();
      for (auto &e : g[u]) {
        int v = e.to;
        if (d[v] == -1 && e.flow < e.w) d[v] = d[u] + 1, q.push(v);
      }
    } return d[t] != -1;
  }
  long long dfs(int u, long long flow) {
    if (u == t) return flow;
    for (int &i = done[u]; i < (int)g[u].size(); i++) {
      edge &e = g[u][i]; if (e.w <= e.flow) continue;
      int v = e.to;
      if (d[v] == d[u] + 1) {
```

```
        long long nw = dfs(v, min(flow, e.w - e.flow));
        if (nw > 0) {
          e.flow += nw;g[v][e.rev].flow -= nw;
          return nw;
        }
      }
    }return 0;
  }
  long long max_flow(int _s, int _t) {
    s = _s;t = _t;long long flow = 0;
    while (bfs()) {
      done.assign(n, 0);
      while (long long nw = dfs(s, inf)) flow += nw;
    }
    flow_through.assign(mxid + 10, 0);
    for(int i = 0; i < n; i++) for(auto e : g[i]) {
        if(e.id >= 0) flow_through[e.id] = e.flow;
    } return flow;
  }
  void reset_flows() {
    for (int i = 0; i < n; ++i) {
      for (auto& e : g[i]) {
        e.flow = 0;
      }
    }
  }
};
```

Mathematics

Sieve

```
// use bitset<N> is_prime; to have O(N/64) memory complexity
// using bitset you can solve upto around N = 10^8 in 1s
const int N = 1e8;vector<int> primes;bitset<N>is_prime;
void sieve_v0() {
  for (int i = 3; i < N; i += 2) {
    is_prime[i] = 1;
  }
  for (int i = 3; i * i < N; i += 2) {
    if (is_prime[i]) {
      for (int j = i * i; j < N; j += i) {
        is_prime[j] = 0;
      }
    }
  }
  is_prime[2] = 1;
  for (int i = 2; i < N; i++) {
    if(is_prime[i]){ primes.push_back(i); }
  }
} // sieve with smallest prime factors (spf)
int spf[N];
void sieve() {
  for (int i = 2; i < N; i++) { spf[i] = i; }
  for (int i = 2; i * i < N; i++) {
    if (spf[i] == i) {
      for (int j = i * i; j < N; j += i) {
        spf[j] = min(spf[j], i);
      }
    }
  }
  for (int i = 2; i < N; i++) {
    if (spf[i] == i) {
      primes.push_back(i);
    }
  }
}
sum of divisor = ((x^(a-1)/(x - 1)) * (y ^ (b-1)/ (y-1)));
```

Euler Totient

```
int phi[N];bool mark[N];
void sieve(){
    for(int i = 1; i < N; i++){ phi[i] = i; }
    for(int i = 2; i < N; i++){
        if(mark[i])continue; phi[i] = i - 1;
        for(int j = i + i; j < N; j += i){
            mark[j] = true; phi[j] -= phi[j] / i;
        }
    }
}
long long euler_phi(long long x){
  long long ans = x;
  for (long long i = 2; i * i <= x; i++){
    if (x % i == 0){
      ans /= i; ans *= i - 1;
      while (x % i == 0){ x /= i; }
    }
  }
  if (x > 1){
    ans /= x;ans *= x - 1;
  }
  return ans;
}
1-> f(n) = gcd(1,n)+gcd(2,n)+...+gcd(n,n)
ans,f(n) = sum{divisor of n as d} d * phi(n/d)
```

Extended Euclid

```
struct triplate { int x,y,gcd;};
//! ax + by = gcd(a,b)
```

```
triplate extendedEuclid(int a, int b) {
  if(b == 0)return {1, 0, a};
  triplate smallans = extendedEuclid(b, a % b);
  triplate ans; ans.x = smallans.y; ans.gcd = smallans.gcd;
  ans.y = (smallans.x - (a / b) * smallans.y);
  return ans;
}// x^-1 = power(x,phi(mod)-1,mod) power(a,b,mod) = (a ^ (b%phi(mod)))%mod;
```

Mobius

```
void mobius() { mu[1] = 1;
  for(int i = 2; i <= N; i++) {
    if(!isComposite[i]) {
      primes.push_back(i);
      mu[i] = -1;
    }
    for(int p : primes) {
      if(1LL * i * p > N)break;
      isComposite[i * p] = 1;
      if(i % p == 0) {mu[i * p] = 0;break;
      }
      else {mu[i * p] = -mu[i];}
    }
  }
}
1-> Number of unordered pairs(i, j) with gcd(a_i, a_j) = 1 freq[x] = how many array elements equal x
cnt[d] = number of array elements divisible by d,pair count = sum{1 to (mx of ar) as d} mu[d] * nCr(cnt[d],2)
 subsequence count = sum{1 to mx} mu[d] * (pow(2,cnt[d])-1)
2->How many pair range[1..N] gcd(x,y) = 1 ,ans = sum{1 to N as d} mu[d]* (N/d)^2
3-> Count Numbers <= N that are divisable by none of primes
L = lcm of all prime.. divs = all divisor of L,ans = sum{divs as d} mu[d]*(N/d)
4-> How many squre free num in [1..N] count = sum{1 to sqrt(N) as d} mu[d]*(N/(d*d))
5-> how many Lattice point with gcd = 1 or 1 <= x <= n and 1 <= y <= m and gcd(x,y) = 1
 ,ans = sum{1 to min(x,y) as d} mu[d]*(n/d)*(m/d)
```

Inclusion-Exclusion

```
int ans = n,la = (1 << k);
for(int i = 1; i < la; i++) { int lc = 1;
  for(int j = 0; j < k; j++) {
    if(getbit(i,j)) { lc = lcm(lc,ar[j]);if(lc > n)break;
    }
  } int cnt = __builtin_popcountll(i);
  if(cnt & 1)ans -= (n / lc);
  else ans += (n / lc);
}
```

Derangements find n

```
int Derangements(int n) {// O(n)
  vector<int> dp(n + 1); dp[1] = 0; dp[2] = 1;
  for (int i = 3; i <= n; i++) {
    dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]);
  }return dp[n];
}
int find_n(int x) {// n*(n+1)/2 = x
  int n = (-1 + sqrt(1 + 8 * x)) / 2;
  if((n * (n + 1)) / 2 > x) { n--;}
  return n;
}
```

Baby-step Giant-step

```
//Returns minimum x for which a ^ x % m = b % m
int baby_step(int a,int b,int m) {
  a%=m,b%=m;int k=1,add=0,g;
  while((g=__gcd(a,m))>1) {
    if(b==k)return add;if(b%g)return -1;
    b/=g,m/=g,add++;k=(k*a/g)%m;
  }
  int n=sqrt(m)+1;int mul=1;
  for(int i=0; i<n; i++)mul=(mul*a)%m;
  unordered_map<int,int>val;
  for(int q=0,cur=b; q<=n; q++) {
    val[cur]=q;cur=(cur*a)%m;
  }
  for(int p=1,cur=k; p<=n; p++) {
    cur=(cur*mul)%m;
    if(val.count(cur)) {return p*n-val[cur]+add;}
  } return -1;
}
```

BigMod/nCr

```
int power(int a,int b,int mod) {
  ll ans = 1;
  while(b>0) {
    if(b&1) { ans = (ans*a)%mod;}
    a = (a*a)%mod;b >>= 1;
  } return ans;
}
int fact[N];
void Pre() {
  fact[0] = 1;
  for(int i = 1; i < N; i++) { fact[i] = (fact[i-1] * i) % mod; }
}
int nCr(int n,int r) {
  if(n < r)return 0ll; ll up = fact[n];
  int down = (fact[r]*fact[n-r]) % mod;
  down = power(down,mod-2,mod);ll res = (up * down) % mod;
```

```
  return res;
}
Catalan Number Cn = (1/(n+1)*nCr(2n,n); Or Cn = nCr(2*n,n) - nCr(2*n,n-1);
```

Chinese Remainder Theorem

```
int ex_gcd(int a,int b,int &x,int &y) {
  if(b == 0) { x = 1; y = 0; return a;}
  int x1,y1; int ans = ex_gcd(b,a % b,x1,y1);
  x = y1; y = x1 - y1 * (a / b); return ans;
}
int chinese_rem_theorem(int a,int m1,int b,int m2) {
  int p,q; int gc = ex_gcd(m1,m2,p,q);
  return (a*m2*q % (m1*m2) + b*m1*p % (m1*m2)) % (m1*m2);
}
void Goriber_solve() {
  int n; cin >> n; vii a(n);
  for(int i = 0; i < n; i++) { cin >> a[i].F >> a[i].S;}
  int x = chinese_rem_theorem(a[0].F,a[0].S,a[1].F,a[1].S);
  int mm = lcm(a[0].S,a[1].S);
  for(int i = 2; i < n; i++) {
    x = chinese_rem_theorem(x,mm,a[i].F,a[i].S); mm = lcm(mm,a[i].S);
  } cout << x << endl;
}
```

Dynamic Programming

SOS DP

```
const ll N = 20; const ll M = (1 << N);
void add_super(int dp[]){
    for(int bit = 0; bit < N; bit++){
        for(int i = 0; i < M; i++){
            if(i & (1 << bit)){ dp[i] += dp[i^(1<<bit)];}
        }
    }
}
void remove_super(int dp[]){
    for(int bit = 0; bit < N; bit++){
        for(int i = M-1; i >= 0; i--){
            if(i & (1 << bit)){ dp[i] -= dp[i^(1<<bit)];}
        }
    }
}
void add_sub(int dp[]){
    for(int bit = 0; bit < N; bit++){
        for(int i = M - 1; i >= 0; i--){
            if(i & (1 << bit)){dp[i^(1<<bit)] += dp[i];}
        }
    }
}
void remove_sub(int dp[]){
    for(int bit = 0; bit < N; bit++){
        for(int i = 0; i < M; i++){
            if(i & (1 << bit)){dp[i^(1<<bit)] -= dp[i];}
        }
    }
}
1.number of element y such that x|y=x ->add_super
2.number of element y such that x&y=x ->add_sub
3.number of element y such that x&y=0 ->add_super,cnt(2^bit-1)^x
4.(a1&a2&a4&a5&..&ax) = k -> add_sub(cnt[]),  cnt[i]=pow(2,cnt[i])-1, remove_sub(cnt[]);
```

Digit DP

```
vi digit;int dp[11][100][100][3];
int sol(int i,int m1,int m2,int f) {
  if(i == n) { return m1 == 0 && m2 == 0;}
  int &ret = dp[i][m1][m2][f];if(ret != -1)return ret;
  int res = 0;int lim = (f == 1 ? 9 : digit[i]);
  for(int j = 0; j <= lim; j++) {
    res += sol(i+1,(m1*10 + j) % k, (m2 + j) % k,(f || j < digit[i]));
  }return ret = res;
}
int cal(int val) {
  digit.clear();
  while(val > 0) {
    digit.push_back(val % 10);val /= 10;
  }
  reverse(all(digit));n = sz(digit);mem(dp,-1);
  return sol(0,0,0,0);
}
void solve() {
  cin >> l >> r >> k;if(k > 90)cout << 0 << endl;
  else cout << cal(r) - cal(l-1) << endl;
}
```

Bit Mask DP

```
int f(int mask){
  if(mask == ((1 << n) - 1))return 0; int &ret = dp[mask];
  if(ret != -1)return ret;int res = INF;
  for(int i = 0; i < n; i++) {
    if((mask & (1 << i)) == 0) { int c = cost[i][i];
      for(int j = 0; j < n; j++) {
        if(mask & (1 << j)) { c += cost[i][j];}
      }
      res = min(res,c + f(mask | (1 << i)));
    }
  }ret = res;
}
```

DP Path Print

```
int f(int i,int rem) {
  if(i == n + 1) {if(rem == 0)return 0;return -n;}
  int &ret = dp[i][rem];
  if(ret != -1)return ret;
  return ret = max(1 + f(i+1,(ar[i]+rem) % k),f(i+1,rem));
}
void print(int i,int rem) {
  if(i == n + 1 || ans == 0) { return; }
  if(1 + f(i+1,(ar[i]+rem)%k) == ans) {
    cout << i << " "; ans--;
    print(i+1,(rem + ar[i])%k);
  }
  else print(i+1,rem);
}
```

String Algorithms

Hashing

```
struct rH{
    const int k = 239,mod = 1000000207;string s;int n;vi pw,vl;
    rH(string s): s(s),n(sz(s)),pw(n+1),vl(n+1){ pw[0] = 1,vl[0] = 0;
        for(int i = 0; i < n; i++){
            vl[i+1] = (1LL *vl[i] * k % mod + s[i]) % mod; pw[i+1] = (1LL*pw[i] * k) % mod;
        }
    }
    int h(int l){ return vl[l]; }
    int h(int l,int r){ return (h(r+1) - (1LL*h(l) * pw[r-l+1]) % mod + mod) % mod;}
};
int k = 1201 ,mod = 1000003891;k = 31,mod = 1000003247;rH hs1(s);cin >> l >> r;
l--,r--;int val = hs.h(l,r);int x = (n - l - 1);int y = x - (r - l);int val2 = hs2.h(y,x);
```

KMP

```
vi build_lps(string p) {
  int sz = p.size();vi lps;lps.assign(sz + 1, 0);int j = 0;lps[0] = 0;
  for(int i = 1; i < sz; i++) {
    while(j >= 0 && p[i] != p[j]) {
      if(j >= 1) j = lps[j - 1]; else j = -1;
    }j++;lps[i] = j;
  }return lps;
}vi ans;
void kmp(vi lps, string s, string p) {
  int psz = p.size(), sz = s.size(), j = 0;
  for(int i = 0; i < sz; i++) {
    while(j >= 0 && p[j] != s[i])
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    j++;
    if(j == psz) {j = lps[j - 1];ans.pb(i - psz + 1); }
  }
}
```

Z Algorithm

```
// An element Z[i] of Z array stores length of the longest substring
// starting from str[i] which is also a prefix of str[0..n-1].
// The first entry of Z array is meaning less as complete string is always prefix of itself.Here Z[0]=0.
vector<int> z_function(string s) {
  int n = (int) s.length();vector<int> z(n);
  for(int i = 1, l = 0, r = 0; i < n; ++i){
    if(i <= r){ z[i] = min (r - i + 1, z[i - l]);}
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]){++z[i];}
    if(i + z[i] - 1 > r){ l = i, r = i + z[i] - 1; }
  }return z;
}
```

Manacher

```
struct manacher { vector<int> p;
  void run_manacher(string s) {
    int n = sz(s);p.assign(n, 1);int l = 1,r = 1;
    for(int i = 1; i < n; i++) {
      p[i] = max(0LL,min(r - i,p[l + r - i]));
      while(i + p[i] < n && i - p[i] >= 0 && s[i + p[i]] == s[i - p[i]]) {
        p[i]++;
      }
      if(i + p[i] > r) {
        l = i - p[i];r = i + p[i];
      }
    }
  }
  void build(string s) {
    string t = "#";for (char c : s) t += c, t += "#";run_manacher(t);
  }
  int get_longest(int cen,bool odd) {
    int pos = 2 * cen + 1 + (!odd);return p[pos] - 1;
  }
  bool is_palindrome(int l,int r) {
    if((r - l + 1) <= get_longest((l + r) / 2,l % 2 == r % 2))return 1;
    else return 0;
  }
};manacher m;m.build(s);
```

Suffix Array

```
#define MAX_N 500010
string T; int RA[MAX_N],tempRA[MAX_N],n;
int SA[MAX_N],tempSA[MAX_N],c[MAX_N];int Phi[MAX_N],PLCP[MAX_N],LCP[MAX_N];
void countingSort(int k){
    int i, sum, maxi = max(300, n);memset(c, 0, sizeof c);
    for(i=0;i<n;i++)c[i+k<n?RA[i+k]:0]++;
    for(i = sum = 0; i < maxi; i++){
        int t = c[i];c[i] = sum;sum += t;
    }
    for(i = 0; i < n; i++){tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];}
    for(i = 0; i < n; i++)SA[i]=tempSA[i];
}
void constructSA(){
    int i, k, r;for(i = 0; i < n; i++)RA[i] = T[i];
    for(i = 0; i < n; i++)SA[i] = i;
    for(k = 1; k < n; k <<= 1){
        countingSort(k);countingSort(0);tempRA[SA[0]] = r = 0;
        for(i = 1; i < n; i++){tempRA[SA[i]] =(RA[SA[i]]==RA[SA[i-1]] && RA[SA[i]+k]==RA[SA[i-1]+k])?r:++r;}
        for(i = 0; i < n; i++)RA[i] = tempRA[i]; if(RA[SA[n-1]] == n-1)break;
    }
}
void computeLCP() {
    int i, L;Phi[SA[0]] = -1;for(i = 1; i < n; i++)Phi[SA[i]] = SA[i-1];
    for(i = L = 0; i < n; i++){
        if(Phi[i] == -1) {PLCP[i] = 0;continue;}while(T[i + L] == T[Phi[i] + L])L++;
        PLCP[i] = L;L = max(L-1, 0);
    }for(i = 0; i < n; i++)LCP[i] = PLCP[SA[i]];
}// main function code ->
cin >> T;T.pb('$');n = sz(T);constructSA();computeLCP();
1.Substring Search->Binary Search on Suffix array
2.Counting Substrings->Binary serach SA Upper-Lowerbound
3.Distrinct Substrings->(n*(n+1))/2 - sumOf(LCP)
4.Longest Common Substring->T+'#'+Pat+'$',SAi < n1 && SAj > n1 maxOf LCP
5.Finding Longest Substring that occur >= K time-> LCP K-1 consicutive = k time max is length of Substring
6.Kth smallest substring->suflen = n-SA[i]-1,i=1 to n-1,new_substr=suflen-LCP[i],if new<=k,len=LCP[i]+k,start=SA[i]
```

Geometry

Basic Geometry

```
struct pt {
    dl x, y;pt() : x(0), y(0) {} pt(dl a, dl b) : x(a), y(b) {}
};// Function to calculate the cross product (direction) of vectors (p1p2) and (p1p3)
int direction(const pt &p1, const pt &p2, const pt &p3) {
    return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
}// Function to check if pt p3 lies on the segment p1p2
bool onsegment(const pt &p1, const pt &p2, const pt &p3) {
    return min(p1.x, p2.x) <= p3.x && p3.x <= max(p1.x, p2.x) &&
           min(p1.y, p2.y) <= p3.y && p3.y <= max(p1.y, p2.y);
}
pt rotateCCW(pt a,dl ang){
    pt v = {a.x,a.y};dl c = cos(ang),s = sin(ang);
    pt res = {v.x*c-v.y*s,v.x*s+v.y*c};
    return res;}
pt rotateCCW(pt a,pt b,dl ang){
    pt v = {b.x-a.x,b.y-a.y};// vec from a to b
    dl c = cos(ang),s = sin(ang);// ang is rediun
    pt res = {v.x*c-v.y*s,v.x*s+v.y*c};
    return res;//rotateCW is (-s put); }
pt unit(pt a){
    double val = sqrt(a.x*a.x+a.y*a.y); return pt(a.x/val,a.y/val); }
pt vec(pt a, pt b){
    return {b.x - a.x, b.y - a.y};}
dl sqnorm(pt a){
    return (a.x*a.x + a.y*a.y);}
//given point a,b and angle abc and len of bc then find point c
pt find_point(pt a,pt b,dl ang,dl len){
    pt c = vec(b,a); c = unit(c);
    c.x *= len, c.y *= len;
    c = rotateCCW(c,ang);
    c.x += b.x, c.y += b.y;
    return c;}
pt find_point(pt b,dl ang,dl len){//same as up
    pt c = {b.x + len*cos(ang),b.y + len*sin(ang)};
    return c;// here ang is CCW , if CW it PI - ang}
dl dot(pt a, pt b){
    return (a.x*b.x + a.y*b.y);}
dl angle(pt a,pt o,pt b){
    pt oa = vec(o , a), ob = vec(o , b);
    if(sqnorm(oa) == 0 || sqnorm(ob) == 0){
        return 0;
    }
    double alpha = acos(dot(oa, ob) / sqrt(sqnorm(oa) * sqnorm(ob)));
    alpha = alpha * 180 / PI;//angle in degree
    return alpha;// angle of AOB}
double abs(pt a) {return sqrt(sqnorm(a));}
int triArea2(const pt &a, const pt &b, const pt &c) {
    return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
}
dl sqDist(const pt &a, const pt &b) {
    return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
dl angle(dl a, dl b,dl c){
    dl alpha = acos((b*b-c*c-a*a)/(-2*a*c));//alpha = alpha*180.0/PI;//angle (ABC) in degree
    return alpha;}
dl Slope(pt a, pt b){
    if(a.x == b.x && a.y == b.y)return oo;
    double dx = b.x - a.x;
    double dy = b.y - a.y;
    if(dx == 0)return oo;
    return (double)dy / dx;}
Basic  L a w > circle arc area = 0.5*r*r*theta ,radius of incircle r = A/s
```

```
radius of circumcircle R=(a*b*c)/(4*A),a spherical cap with height H and R = H/2
dl volume = (PI * h * h/ 3.0) * (3.0 * R - h);Polygone n point and same distance s
dl area_polygone=(n*s*s)/(4*(tan(PI/n)));dl redius_polygone = s / (2*sin(PI / n));
glass area = ((pi*h)/3) *((r1*r1)+(r2*r2)+(r1*r2)) ;N inner circle in One circle R
dl theta = PI / n;dl r = (R * sin(theta)) / (1 + sin(theta));dl area = PI*r*r;
dl E = (R - r);dl AreaNotcoverInner = ((n * r * r) / tan(theta)) - (area * (n - 2)) / 2.0;
dl AreaNotcoverOuter = Area - n * area - AreaNotcoverInner;
Tseq = (2^N - 1) = Even Subsequence + OddSubsequence ;Eseq = 2^Even;
Oseq = 2^(Odd - 1);OddSumSeq = (2^Even * (2^(Odd - 1)));EvenSumSeq = ((2^N) - 1 - OddSumSeq);
Odd= Total number of odd in the array.;Oseq=Total number of subsequences with only odd;
Num ;OddSumSeq = Total number of subsequences with odd sum
```

Convex Hull

```
ConvexHull : Grahams Scan O(n lg n), integer implementation ;P[]: holds all the points, C[]: holds points on the hull
np: number of points in P[], nc: number of points in C[]
to handle duplicate, call makeUnique() before calling convexHull()call convexHull() if you have np >= 3
to remove co-linear points on hull, call compress() after convexHull()
struct point{ int x, y; };
point P[N], C[N], P0;
int triArea2(const point &a, const point &b, const point &c) {
    return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
}
int sqDist(const point &a, const point &b) {
    return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
bool comp(const point &a, const point &b) {
    int d = triArea2(P0, a, b); if(d < 0) return false;
    if(!d && sqDist(P0, a) > sqDist(P0, b)) return false;
    return true;
}
bool normal(const point &a, const point &b) {
    return ((a.x==b.x) ? a.y < b.y : a.x < b.x);
}
bool issame(const point &a, const point &b) {
    return (a.x == b.x && a.y == b.y);
}
void makeUnique(int &np) {
    sort(&P[0], &P[np], normal);
    np = unique(&P[0], &P[np], issame) - P;
}
void convexHull(int &np, int &nc) {
    int i, j, pos = 0;
    for(i = 1; i < np; i++){
        if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x<P[pos].x))pos = i;
    }
    swap(P[0], P[pos]); P0 = P[0]; sort(&P[1], &P[np], comp);
    j = np - 1,i = np - 1;
    while(j > 0 && triArea2(P0, P[j], P[j-1]) == 0)j--;
    while(i > j){
        swap(P[i], P[j]); i--,j++;
    }
    for(i = 0; i < 3; i++) C[i] = P[i];
    for(i = j = 3; i < np; i++) {
        while(triArea2(C[j-2], C[j-1], P[i]) < 0) j--;
        C[j++] = P[i];
    }
    nc = j;
}
void compress(int &nc) {
    int i, j, d;
    C[nc] = C[0];
    for(i=j=1; i < nc; i++) {
        d = triArea2(C[j-1], C[i], C[i+1]);
        if(d || (!d && issame(C[j-1], C[i+1]))) C[j++] = C[i];
    }
    nc = j;
}
```

Circle Intersection

```
dl commonArea(const Circle &a, const Circle &b) {
    int dsq = sqDist(a, b); double d = sqrt((double)dsq);
    if(sq(a.r + b.r) <= dsq) return 0;
    if(a.r >= b.r && sq(a.r-b.r) >= dsq) return PI * b.r * b.r;
    if(a.r <= b.r && sq(b.r-a.r) >= dsq) return PI * a.r * a.r;
    double angleA = 2.0 * acos((a.r * a.r + dsq - b.r * b.r) / (2.0 * a.r * d));
    double angleB = 2.0 * acos((b.r * b.r + dsq - a.r * a.r) / (2.0 * b.r * d));
    return 0.5 * (a.r * a.r * (angleA - sin(angleA)) + b.r * b.r * (angleB - sin(angleB)));
}
```

Point in Polygon

```
bool inConvexPoly(int nc, const pt &p) {
    int st = 1, en = nc - 1, mid;
    while(en - st > 1) {
        mid = (st + en)>>1;
        if(direction(C[0], C[mid], p) < 0) en = mid;
        else st = mid;
    }
    if(direction(C[0], C[st], p) < 0) return false;
    if(direction(C[st], C[en], p) < 0) return false;
    if(direction(C[en], C[0], p) < 0) return false;
    return true;
}//Ray casting algo to check point in convex -> start here
for(int i = 0; i < n; ++i){// Check if the pt is on any boundary edge
  pt a = polygon[i];
  pt b = polygon[(i + 1) % n];
  if(onSegment(a, b, p)) {return "BOUNDARY";}
}// Ray casting algorithm ;int count = 0;
for(int i = 0; i < n; ++i) {
  pt a = polygon[i];pt b = polygon[(i + 1) % n];
  if ((a.y > p.y) != (b.y > p.y)) {// Check if the pt is between the y-coordinates of the edge
```

```
    // Calculate the x-intersection of the edge with the horizontal ray
    double xIntersect = (double)(b.x - a.x) * (p.y - a.y) / (b.y - a.y) + a.x;
    if(p.x <= xIntersect) {count++;}
  }
}if(count % 2 == 1){return "INSIDE";}else {return "OUTSIDE";}
```

Segment Intersection

```
struct pt {// Segment Intersection check
    int x, y; pt() : x(0), y(0) {} pt(int a, int b) : x(a), y(b) {}
};
// Function to calculate the cross product (direction) of vectors (p1p2) and (p1p3)
int direction(const pt &p1, const pt &p2, const pt &p3) {
    return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
}
// Function to check if pt p3 lies on the segment p1p2
bool onsegment(const pt &p1, const pt &p2, const pt &p3) {
    return min(p1.x, p2.x) <= p3.x && p3.x <= max(p1.x, p2.x) && min(p1.y, p2.y) <= p3.y && p3.y <= max(p1.y, p2.y);
}
// Function to check if two segments (p1p2) and (p3p4) intersect
inline bool intersect(const pt &p1, const pt &p2, const pt &p3, const pt &p4) {
    int d1, d2, d3, d4;
    d1 = direction(p3, p4, p1);  // Direction of (p3, p4, p1)
    d2 = direction(p3, p4, p2);  // Direction of (p3, p4, p2)
    d3 = direction(p1, p2, p3);  // Direction of (p1, p2, p3)
    d4 = direction(p1, p2, p4);  // Direction of (p1, p2, p4)
    // General case: Segments (p1p2) and (p3p4) intersect if directions differ
    if (((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) && ((d3 < 0 && d4 > 0) || (d3 > 0 && d4 < 0)))return true;
    // Special cases: Check if the pts are collinear and lie on the segment
    if (!d3 && onsegment(p1, p2, p3)) return true;  // p3 is on segment (p1p2)
    if (!d4 && onsegment(p1, p2, p4)) return true;  // p4 is on segment (p1p2)
    if (!d1 && onsegment(p3, p4, p1)) return true;  // p1 is on segment (p3p4)
    if (!d2 && onsegment(p3, p4, p2)) return true;  // p2 is on segment (p3p4)
    return false;
}//Segment intersection point start here
pair<bool, pt> doSegmentsIntersect(const pt &p1,const pt &p2,const pt &p3,const pt &p4) {
    dl d1 = direction(p3, p4, p1);dl d2 = direction(p3, p4, p2);
    dl d3 = direction(p1, p2, p3);dl d4 = direction(p1, p2, p4);
    if (d1 * d2 < 0 && d3 * d4 < 0) {// Cramer's Rule.
        dl det = (p1.x - p2.x) * (p3.y - p4.y) - (p1.y - p2.y) * (p3.x - p4.x);
        dl ix = ((p1.x*p2.y - p1.y*p2.x)*(p3.x - p4.x) - (p1.x - p2.x)*(p3.x*p4.y - p3.y*p4.x)) / det;
        dl iy = ((p1.x*p2.y - p1.y*p2.x)*(p3.y - p4.y) - (p1.y - p2.y)*(p3.x*p4.y - p3.y*p4.x)) / det;
        return {true, pt(ix, iy)};
    }// Special cases - endpts on other segment
    if (d1 == 0 && onSegment(p3, p4, p1)) return {true, p1};
    if (d2 == 0 && onSegment(p3, p4, p2)) return {true, p2};
    if (d3 == 0 && onSegment(p1, p2, p3)) return {true, p3};
    if (d4 == 0 && onSegment(p1, p2, p4)) return {true, p4};
    return {false, pt(0,0)};
}
```

Polygon Area

```
int area = 0,b = 0;
for(int i = 0; i < n; i++){
    Point l = p[i], r = p[i+1];area += (l.x * r.y - r.x * l.y);l.x -= r.x, l.y -= r.y;
    int g = gcd(l.x, l.y);b += abs(g);}
peaks theorem  a = abs(area) - b + 2;area = abs(area) / 2;// same same but
A=I+2 B 1 , A = area of polygone,I = inside point,B = Boundary point;p[n] = p[0];int area = 0,b = 0;
for(int i = 0; i < n; i++){
    Point l = p[i], r = p[i+1];area += (l.x * r.y - r.x * l.y);
    l.x -= r.x, l.y -= r.y;int g = gcd(l.x, l.y);b += abs(g);
}int a = abs(area) - b + 2;
```

Distance from Point to line Segment

```
// Function to calculate the distance from point C to line segment AB
double linePointDist(int A[], int B[], int C[], bool isSegment) {
    int ABx = B[0] - A[0], ABy = B[1] - A[1];int ACx = C[0] - A[0], ACy = C[1] - A[1];
    int BCx = C[0] - B[0], BCy = C[1] - B[1];double crossProd = ABx * ACy - ABy * ACx;
    double ABLength = sqrt(ABx * ABx + ABy * ABy);double dist = fabs(crossProd) / ABLength;
    if (isSegment) {
        if ((ABx * BCx + ABy * BCy) > 0) return sqrt(BCx * BCx + BCy * BCy);  // Distance to B
        if ((-ABx * ACx - ABy * ACy) > 0) return sqrt(ACx * ACx + ACy * ACy);  // Distance to A
    } return abs(dist); }
```

Circle from 3 Points

```
// Function to find the center of the circle given 3 points
pair<dl, dl> findCircleCenter(dl x1, dl y1, dl x2, dl y2, dl x3, dl y3) {
    dl mx1 = (x1 + x2) / 2, my1 = (y1 + y2) / 2, mx2 = (x2 + x3) / 2, my2 = (y2 + y3) / 2;
    dl A1 = y2 - y1, B1 = x1 - x2, A2 = y3 - y2, B2 = x2 - x3; dl det = A1 * B2 - A2 * B1;
    dl cx = (B2 * (A1 * mx1 - A2 * mx2) - B1 * (A1 * mx2 - A2 * mx1)) / det;
    dl cy = (A1 * (B2 * my2 - B1 * my1) - A2 * (B1 * my1 - B2 * my2)) / det;
    return {cx, cy};
}
```

Advanced Algorithms

Heavy-Light Decomposition

```
const ll N = 1e5 + 123;const ll M = 4*N + 5;vi graph[N];ll tree[M];
int Sz[N],heavy[N],Par[N],lev[N]; int Head[N],pos[N],idx = 1,ar[N],val[N];
void dfs(int node,int pa){
    Sz[node] = 1; Par[node] = pa;
    for(auto u : graph[node]){
        if(u == pa)continue;
        lev[u] = lev[node] + 1;dfs(u,node);Sz[node] += Sz[u];
```

```
        if(Sz[u] > Sz[heavy[node]] || heavy[node] == 0){
            heavy[node] = u;
        }
    }
}
void dfsHLD(int node,int chain){
    Head[node] = chain;ar[idx] = val[node];pos[node] = idx;idx++;
    if(heavy[node] != 0){
        dfsHLD(heavy[node],chain);
    }
    for(auto u : graph[node]){
        if(u == Par[node])continue;
        if(heavy[node] != u){ dfsHLD(u,u);}
    }
}
int Query(int a,int b){
    int mx = 0;
    while(Head[a] != Head[b]){
        if(lev[Head[a]] < lev[Head[b]]){ swap(a,b);}
        mx = max(mx,query(1,1,idx,pos[Head[a]],pos[a]));
        a = Par[Head[a]];
    }
    if(lev[a] < lev[b])swap(a,b);int l = pos[b]+1,r = pos[a];
    if(l <= r)mx = max(mx,query(1,1,idx,l,r));//edge base b+1 to a,node base b to a
    return mx;
}// main function start->
int n,q;cin >> n;
for(int i = 1; i <= n; i++){
    graph[i].clear();Sz[i] = 0;lev[i] = 0;heavy[i] = 0;pos[i] = 0,val[i] = 0;Head[i] = 0,Par[i] = 0;
}
map<int,pair<int,int>> bd;vector<tu> edge;
for(int i = 1; i < n; i++){
    int u,v,w;cin >> u >> v >> w;graph[u].push_back(v);graph[v].push_back(u);
    bd[i] = {u,v};edge.push_back({u,v,w});}
idx = 1;dfs(1,0);val[1] = 0;
for(auto u : edge){int a = u[0];if(lev[u[0]] < lev[u[1]])a = u[1];val[a] = u[2];}
dfsHLD(1,1);init(1,1,idx);
while(1){string s;cin >> s;if(s == "DONE")break;
    if(s == "CHANGE"){
        int i,v;cin >> i >> v;pair<int,int> x = bd[i];int a = x.F;
        if(lev[x.F] < lev[x.S])a = x.S;int del = pos[a];upgrade(1,1,idx,del,v);}
    else{int a,b;cin >> a >> b;cout << Query(a,b) << endl;}
}
```

Mo's Algorithm

```
int ar[N],ans,n,answers[N],cnt[N];int block_size;
void remove(int idx){ cnt[ar[idx]]--; if(cnt[ar[idx]] == 0)ans--;}
void add(int idx){ cnt[ar[idx]]++; if(cnt[ar[idx]] == 1)ans++;}
struct Query{
    int l, r, idx;
    bool operator < (const Query &x) const {
        if(l / block_size == x.l / block_size) return r > x.r;
        return l / block_size < x.l / block_size;}
};// main function
void Goriber_solve(){cin >> n; for(int i = 0; i < n; i++){cin >> ar[i];}int q;cin >> q;
    block_size =  (int) sqrt (n + .0) + 1;vector<Query> queries;
    for(int i = 0; i < q; i++){int l,r;cin >> l >> r;queries.pb({l,r,i});}
    sort(all(queries));int cur_l = 0;int cur_r = -1;
    for (Query &q : queries) {q.l--, q.r--;
        while (cur_l > q.l) {cur_l--;add(cur_l);}
        while (cur_r < q.r) {cur_r++;add(cur_r);}
        while (cur_l < q.l) {remove(cur_l);cur_l++;}
        while (cur_r > q.r) {remove(cur_r);cur_r--;}
        answers[q.idx] = ans;}
    for(int i = 0; i < q; i++){
        cout << answers[i] << endl;}}
```

Square Root Decomposition

```
int n;vector<int> a (n);int len = (int) sqrt (n + .0) + 1;vector<int> b (len);
for(int i=0; i<n; ++i) b[i / len] += a[i];
while(q--) {
  int l, r;int sum = 0;
  for(int i=l; i<=r; ){
    if(i % len == 0 && i + len - 1 <= r) {
      sum += b[i / len];i += len;}
    else {sum += a[i];++i;}
  }
}
```

Euler Tour

```
vector<int> graph[N];vector<int> flat_tree;map<int,pair<int,int>> mp;int cnt = 0;
void dfs(int node,int par) {
  flat_tree.push_back(node);mp[node].first = cnt;cnt++;
  for(auto child : graph[node]) {
    if(child == par)continue;dfs(child,node);
  }
  mp[node].second = cnt;flat_tree.push_back(node);cnt++;}
```

Tree Diameter

```
vi graph[N];
void dfs(int node,int par,int lev,vi &d){ d[node] = lev;
    for(auto u : graph[node]){
        if(u != par){ dfs(u,node,lev+1,d);}
    }
}
void Goriber_solve(){
    int n;cin >> n;for(int i = 0; i <= n; i++){graph[i].clear();}
```

```
    for(int i = 1; i < n; i++){
        int u,v;cin >> u >> v;u--,v--;graph[u].pb(v);graph[v].pb(u);
    }
    vi d1(n,0),d2(n,0);dfs(0,-1,0,d1);int a = max_element(all(d1)) - d1.begin();dfs(a,-1,0,d1);
    int b = max_element(all(d1)) - d1.begin();dfs(b,-1,0,d2);}
```

2D Algorithms

2D Pre Sum + mat Rotation + grid 8 direction move

```
pre[i][j] = ar[i][j] + pre[i-1][j] + pre[i][j-1] - pre[i-1][j-1];
cout<< pre[c][d] - pre[a-1][d] - pre[c][b-1] + pre[a-1][b-1] ;
for(int i = 0; i < n; i++) {
  for(int j = 0; j < n; j++) {res[j][n - i - 1] = mat[i][j];}
}
int fx[]= {+0,+0,+1,-1,-1,+1,-1,+1};int fy[]= {-1,+1,+0,+0,+1,+1,-1,-1};
```

Spiral Matrix

```
int x = (n + 1) / 2, y = (n + 1) / 2;int num = 0;ans[x][y] = 0;// 1 based matrix
for (int i = 2; i <= n; ++i) {
  if (i % 2 == 0) { ans[x][++y] = ++num;
    for (int j = 1; j < i; ++j)ans[++x][y] = ++num;
    for (int j = 1; j < i; ++j)ans[x][--y] = ++num;
  }
  else {ans[x][--y] = ++num;
    for (int j = 1; j < i; ++j)ans[--x][y] = ++num;
    for (int j = 1; j < i; ++j)ans[x][++y] = ++num;
}   }
```

Matrix Exponentiation

```
vector<vi>matMul(vector<vi>&mat1,vector<vi>&mat2,int n){ vector<vi> res(n, vi(n,0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
        for(int k = 0; k < n; k++){res[i][j]=(res[i][j]+((ll)mat1[i][k]*mat2[k][j])%mod)%mod;}
        }
    } return res;
}
vector<vi>matExpo(vector<vi>& mat,int k,int n){
    if(k == 1)return mat; vector<vi>res=matExpo(mat,k/2,n); res =matMul(res,res,n);
    if(k & 1)res=matMul(res,mat,n);
    return res;
}
F(k+1) =|mat^k|F(k)  |
F(k) =  |     |F( k 1 )|
S(k) =  |     |S( k 1 )|
S( k 1 ) =|     |S( k 2 )|
vi mulBase(vector<vi>&mat,vi &base,int n){ vi res(n,0);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            res[i] += (ll)mat[i][j] * base[j] % mod;res[i] %= mod;
        }
    } return res;
}
```

Utilities

Path Compression

```
vi a(n), b(n);sort(all(b));b.erase(unique(all(b)), b.end());
for(int i = 0; i < n; i++){a[i] = lower_bound(all(b), a[i]) - b.begin() + 1;}
```

Histogram

```
int getMaxArea(vector<int>&arr, int n) {
  stack<int> s;s.push(-1);int area = arr[0];int i = 0;vector<int> left(n, -1), right(n, n);
  while(i < n) {
    while(!s.empty() && s.top() != -1 && arr[s.top()] > arr[i]) {
      right[s.top()] = i;s.pop();
    }
    if(i > 0 && arr[i] == arr[i - 1]) {
      left[i] = left[i - 1];
    }else { leftr[i] = s.top();}
    s.push(i);i++;
  }
  for (int j = 0; j < n; j++) {
    area = max(area, arr[j] * (right[j] - left[j] - 1));
  } return area;
}
```

Is Intersect All Points

```
auto f = [&] (dl m) -> bool {vector<pair<dl,dl>> p;
  for(int i = 0; i < n; i++) {
    dl r = x[i] + m*v[i];dl l = x[i] - m*v[i];p.push_back({l,r});
  }
  dl l = -INF,r = INF;
  for(int i = 0; i < n; i++) {
    dl left = p[i].F;dl right = p[i].S;
    if(l > right || r < left) {return 0;} l = max(l,left); r = min(r,right);
  } return 1;
};
```

Unique Line Equations

```
tuple<int,int,int> norm(int a, int b, int c) {
  int d = (__gcd(a,__gcd(b,c))); a/=d; b/=d; c/=d;int s = a?a:b?b:c;
  if(s < 0) a=-a, b=-b, c=-c;
  return {a,b,c};
}
int countLines(vector<pair<int,int>>& p) {
  set<tuple<int,int,int>> s;
  for(int i=0; i<p.size(); i++) {
    for(int j=i+1; j<p.size(); j++) {
      auto [x1,y1]=p[i], [x2,y2]=p[j]; int a=y2-y1, b=x1-x2, c=x2*y1-x1*y2; s.insert(norm(a,b,c));
    }
  }return s.size();
}
```

Special Problems

Kth Number L to R Using Persistent Seg Tree

```
//number of elements greater than k in the subsequence ai, ai+1 ... aj
struct node{
    int val;
    node* lc;
    node* rc;
};
int ar[N];
node* versions[N+5];
vector<node*> lagbe;

node* build(int l,int r){
    if(l == r){
        node* me = new node();
        lagbe.pb(me);
        me->val = 0;
        me->lc = NULL;
        me->rc = NULL;
        return me;
    }
    node* my = new node();
    lagbe.pb(my);
    int mid = (l + r)/2;
    my->lc = build(l,mid);
    my->rc = build(mid+1,r);
    my->val = (my->lc->val + my->rc->val);
    return my;
}
int query(node* n1,node* n2,int l,int r,int L,int R){
    if(n1 == NULL || n2 == NULL)return 0;
    if(R < l || r < L)return 0;
    if(L <= l && r <= R){
        return n1->val - n2->val;
    }
    int mid = (l + r)/2;
    int p = query(n1->lc,n2->lc,l,mid,L,R);
    int q = query(n1->rc,n2->rc,mid+1,r,L,R);
    return p + q;
}
node* update(node* my,int l,int r,int ind,int val){
    if(l == r){
        node* new_my = new node();
        lagbe.pb(new_my);
        new_my->lc = NULL;
        new_my->rc = NULL;
        new_my->val = my -> val + val;
        return new_my;
    }
    node* new_me = new node();
    lagbe.pb(new_me);
    int mid = (l + r)/2;
    if(ind <= mid){
        node* x = update(my->lc,l,mid,ind,val);
        new_me->lc = x;
        new_me->rc = my->rc;
    }
    else{
        node* y = update(my->rc,mid+1,r,ind,val);
        new_me->lc = my->lc;
        new_me->rc = y;
    }
    new_me->val = new_me->lc->val + new_me->rc->val;
    return new_me;
}
vi com;
void Goriber_solve()
{
    int n,q;
    cin >> n;
    for(int i = 1; i <= n; i++){
        cin >> ar[i];
        com.pb(ar[i]);
    }
    sort(all(com));
    com.resize(unique(all(com))-com.begin());
    map<int,int> pos;
    int x = 0;
    for(auto u : com)pos[u] = x++;
    int si = sz(com)+1;
    versions[0] = build(0,si);
    for(int i = 1; i <= n; i++){
        versions[i] = update(versions[i-1],0,si,pos[ar[i]],1);
    }
    int ans = 0;
    cin >> q;
```

```
    while(q--){
        int a,b,c;
        cin >> a >> b >> c;
        a = (a ^ ans);
        b = (b ^ ans);
        c = (c ^ ans);
        int val = upper_bound(all(com),c)-com.begin();
        if(a <= 0)a = 1;
        if(b > n)b = n;
        if(a > b){
            cout << 0 << endl;
            ans = 0;
            continue;
        }
        ans = query(versions[b],versions[a-1],0,si,val,si);
        cout << ans << endl;
    }
    for(auto u : lagbe){
        delete u;
    }
}
```

How Many Subarrays XOR $< k$

```
class Trie{
    int cnt(int num, int k){
        Node* cur = root;int ret = 0;
        for(int i = 31; i >= 0; i--){
            int num1 = (num >> i) & 1;int num2 = (k >> i) & 1;
            if(num2){
                if(cur -> contains(num1)){ ret += cur -> get(num1) -> getPre();}
                if(cur -> contains(num1 ^ 1))cur = cur -> get(num1 ^ 1);
                else return ret;
            }
            else{
                if(cur -> contains(num1))cur = cur -> get(num1);
                else return ret;
            }
        }
        return ret;
    }
};// main function trie.insert(0); xr ^= ar[i]; ans += trie.cnt(xr,k);trie.insert(xr);
```

Min Length with Max XOR $\geq k$

```
class Trie {
  int getMax(int num,int lo) {
    Node* cur_node = root;int mx = 0;
    for(int i = 31; i >= 0; i--) {
      int bit = (num >> i) & 1;int opo = bit ^ 1;
      if(cur_node -> contains(opo) && cur_node -> get(opo) -> mx_index >= lo) {
        mx = ((1LL << i) | mx);cur_node = cur_node -> get(opo);
      }
      else {cur_node = cur_node -> get(bit);}
    }
    return mx;
  }
};// main function -> int n,k; cin >> n >> k; int ans = n + 1; Trie trie;
for(int i = 1; i <= n; i++) { int x; cin >> x; trie.insert(x,i); int lo = 1, hi = i,cnt = -1;
  while(lo <= hi) { int mid = (lo + hi) / 2;
    if((trie.getMax(x,mid)) >= k) { cnt = mid; lo = mid + 1;}
    else hi = mid - 1;
  } if(cnt != -1)ans = min(ans,i + 1 - cnt);
} if(ans == n + 1)cout << -1 << endl; else cout << ans << endl;
```

$I < J < K \quad and \quad a_i > a_j > a_k$

```
vi a(n), b(n); for(int i = 0; i < n; i++) { cin >> a[i]; b[i] = a[i]; }
sort(all(b)); b.erase(unique(all(b)), b.end());
for(int i = 0; i < n; i++) { a[i] = lower_bound(all(b), a[i]) - b.begin() + 1; } vi L(n), R(n);
for(int i = 0; i < n; i++) {
  L[i] = sum(a[i]+1,n+1); update(a[i],1); mem(tree,0);// use BIT}
  for(int i = n-1; i >= 0; i--) {
    R[i] = sum(1,a[i]-1); update(a[i],1);
  } int ans = 0;
  for(int i = 0; i < n; i++) {
    ans += L[i] * R[i];
  } cout << ans << endl;}
```

Tetrahedron Formulas

```
glass area = ((pi*h)/3) *((r1*r1)+(r2*r2)+(r1*r2)) //Some tetrahedron formulas
inline double volume(double u, double v, double w, double U, double V, double W) {
    double u1,v1,w1;  u1 = v * v + w * w - U * U;
    v1 = w * w + u * u - V * V;  w1 = u * u + v * v - W * W;
    return sqrt(4.0*u*u*v*v*w*w - u*u*u1*u1 - v*v*v1*v1 - w*w*w1*w1 + u1*v1*w1) / 12.0; }
double surface(double a, double b, double c) {
    return sqrt((a + b + c) * (-a + b + c) * (a - b + c) * (a + b - c)) / 4.0}
double insphere(double WX, double WY, double WZ, double XY, double XZ, double YZ) {
    double sur, rad;
    sur = surface(WX, WY, XY) + surface(WX, XZ, WZ) + surface(WY, YZ, WZ) + surface(XY, XZ, YZ);
    rad = volume(WX, WY, WZ, YZ, XZ, XY) * 3.0 / sur;   return rad;}
```

Templates

Saad Segment Tree

```
struct ST {
```

```
    struct Node { // change here
        long long sum = 0; Node () {}
    };
    vector<long long> a;
    vector<Node> t;
    Node op (Node x, Node y) {          // change here
        x.sum += y.sum;
        return x;
    }
    ST (vector<long long> &A) {         // change here
        a = A;
        t.resize (4 * a.size());
        bld (1, 0, a.size() - 1);
    }
    void bld (int i, int l, int r) {
        if (l == r) {
            t[i].sum = a[l];
            return;
        }
        int m = (l + r) / 2;
        bld (i * 2, l, m);
        bld (i * 2 + 1, m + 1, r);
        t[i] = op (t[i * 2], t[i * 2 + 1]);
    }
    void upd (int i, int l, int r, int p, int v) {
        if (l == r) {
            a[p] += v;                   // change here
            t[i].sum = a[p];
            return;
        }
        int m = (l + r) / 2;
        if (p <= m) upd (i * 2, l, m, p, v);
        else upd (i * 2 + 1, m + 1, r, p, v);
        t[i] = op (t[i * 2], t[i * 2 + 1]);
    }
    Node qry (int i, int l, int r, int L, int R) {
        if (R < l || r < L) return Node();
        if (L <= l && r <= R) return t[i];
        int m = (l + r) / 2;
        return op (qry (i * 2, l, m, L, R),
                   qry (i * 2 + 1, m + 1, r, L, R));
    }
};
struct ST {
    struct N {
        long long s = 0, m = 1e18;
    };
    vector<N> t;
    vector<long long> a, lz;
    int n;
    N op (N x, N y) {
        x.s += y.s;
        x.m = min (x.m, y.m);
        return x;
    }
    void push (int i, int l, int r, long long v) {
        t[i].s = v * (r - l + 1);
        t[i].m = v;
        if (l != r) lz[i * 2] = lz[i * 2 + 1] = v;
        lz[i] = 0;
    }
    void build (int i, int l, int r) {
        if (l == r) {
            t[i].s = a[l];
            t[i].m = a[l];
            return;
        }
        int m = (l + r) / 2;
        build (i * 2, l, m);
        build (i * 2 + 1, m + 1, r);
        t[i] = op (t[i * 2], t[i * 2 + 1]);
    }
    void upd (int i, int l, int r, int L, int R, long long v) {
        if (lz[i]) push (i, l, r, lz[i]);
        if (r < L || R < l) return;

        if (L <= l && r <= R) {
            push (i, l, r, v);
            return;
        }

        int m = (l + r) / 2;
        upd (i * 2, l, m, L, R, v);
        upd (i * 2 + 1, m + 1, r, L, R, v);
        t[i] = op (t[i * 2], t[i * 2 + 1]);
    }
    N qry (int i, int l, int r, int L, int R) {
        if (r < L || R < l) return N();
        if (lz[i]) push (i, l, r, lz[i]);
        if (L <= l && r <= R) return t[i];

        int m = (l + r) / 2;
        return op (qry (i * 2, l, m, L, R), qry (i * 2 + 1, m + 1, r, L, R));
    }
    ST (vector<long long> &A) {
        a = A;
        n = A.size();
        t.resize (4 * n);
        lz.assign (4 * n, 0);
        build (1, 0, n - 1);
    }
    void InUpdate (int i, long long v) {
        upd (1, 0, n - 1, i, i, v);
    }

    void R_Update (int L, int R, long long v) {
```

```
        upd (1, 0, n - 1, L, R, v);
    }

    N query (int L, int R) {
        return qry (1, 0, n - 1, L, R);
    }
};
```

Kaium Segment Tree

```
struct Node { // Maximum Subarray Sum
  int sum;
  int sub; // Value of interest
  int pre;
  int suf;

  Node(): sum(0), sub(-1e18), pre(sub), suf(sub) {}
  Node(int x): sum(x), sub(x), pre(x), suf(x) {}
  operator int() { return sub; }

  Node operator+(const Node& b) {
    Node c;
    c.sum = sum + b.sum;
    c.sub = max({ sub, b.sub, suf+b.pre });
    c.pre = max(pre, sum+b.pre);
    c.suf = max(b.suf, suf+b.sum);
    return c;
  }
};

struct SegTree {
  int n;
  vector<Node> Tree;
  SegTree(int size) {
    // Round up to closest power of 2
    n = 1 << (32 - __builtin_clz(--size));
    Tree.resize(2*n);
  }

  // Constructor to initialize by array
  SegTree(const VI& A): SegTree(A.size()) {
    for (int i = 0; i < A.size(); i++)
      Tree[n + i] = Node(A[i]);
    // Build the segment tree
    ROF (i, 1, n-1)
      Tree[i] = Tree[2*i] + Tree[2*i+1];
  }

  // Update value at position p
  void update(int p, int value) {
    p += n;
    Tree[p] = Node(value);
    for (p /= 2; p > 0; p /= 2)
      Tree[p] = Tree[2*p] + Tree[2*p+1];
  }

  // Query on interval [l, r)
  Node query(int l, int r) {
    Node resL, resR;
    for (l += n, r += n; l < r; l /= 2, r /= 2) {
      if (l % 2) resL = resL + Tree[l++];
      if (r % 2) resR = Tree[--r] + resR;
    } // Non-commutative merge
    return resL + resR;
  }
};
```

Kaium I/O

```
#define DEBUG
#define LOCAL
// #define TCASE
// #define PB_DS

#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
#define _LB_ cerr << "{ "
#define _RB_ cerr << " }"
#else
#define endl '\n'
#define _LB_
#define _RB_
#endif

#define _SP_ << &" "[!i++] <<
#define _FN_OUT_ { _LB_; int i = 0; for (auto& a : A) out _SP_ a; _RB_; return out; }
#define _FN_IN_ { for (auto& a : A) in >> a; return in; }
#define _OP_OUT_ ostream& operator<<(ostream& out, const
#define _OP_IN_ istream& operator>>(istream& in,

#define temT template <class T>
#define temTs template <class... Ts>
#define temTN template <class T, size_t N>

temTs _OP_OUT_ pair<Ts...>& p) { _LB_; out << p.first << " " << p.second; _RB_; return out; }
temTs _OP_IN_ pair<Ts...>& p) { return in >> p.first >> p.second; }
temTN _OP_OUT_ array<T, N>& A) _FN_OUT_
temTN _OP_IN_ array<T, N>& A) _FN_IN_
#define def_out(cont) temT _OP_OUT_ cont<T>& A) _FN_OUT_
#define def_in(cont) temT _OP_IN_ cont<T>& A) _FN_IN_
def_in(vector) def_in(deque) def_out(vector) def_out(deque) def_out(set) def_out(multiset)
temTs _OP_OUT_ map<Ts...>& A) _FN_OUT_
```

```

#ifdef PB_DS
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define _RBT_ rb_tree_tag, tree_order_statistics_node_update
temT using ordered_set = tree<T, null_type, less<T>, _RBT_>;
temT using ordered_multiset = tree<T, null_type, less_equal<T>, _RBT_>;
def_out(ordered_set) def_out(ordered_multiset)
#endif

temTs istream& c_in(Ts&... args) { return ((cin >> args), ...); }
temTs ostream& c_out(const Ts&... args) { int i = 0; return ((cout _SP_ args), ...) << endl; }
ostream& c_out(bool b) { return c_out(b ? "YES" : "NO"); }

#ifdef LOCAL
temTs ostream& c_err(const Ts&... args) { int i = 0; return ((cerr _SP_ args), ...) << endl; }
#else
#define c_err(...)
#endif
#define d_bug(args...) c_err(#args, '=', args)


#define all(A) A.begin(), A.end()
#define rall(A) A.rbegin(), A.rend()
#define sum_of(A) accumulate(all(A), 0ll)
#define min_of(A) *min_element(all(A))
#define max_of(A) *max_element(all(A))

#define int long long
#define uint unsigned int
using VI = vector<int>;
using VC = vector<char>;
using DI = deque<int>;
using QI = queue<int>;
using SI = set<int>;
using MII = map<int, int>;
using VVI = vector<VI>;
template<size_t N> using AI = array<int, N>;
using II = AI<2>; // instead of pair<int, int>
using III = AI<3>; // ... tuple<int, int, int>
using VII = vector<II>;
using VIII = vector<III>;
using VVII = vector<VII>;
template<class T> using PQ = priority_queue<T, vector<T>, greater<T>>;
#define VMAT(n, m) VVI(n, VI(m))
#define VMATa(n, m, a) VVI(n, VI(m, a))
#define AMAT(n, m) vector<AI<m>>(n)
#define FOR(i, l, r) for (int i = l; i <= r; i++)
#define ROF(i, l, r) for (int i = r; i >= l; i--)
```

Kaium Hashing

```
II operator+(II a, II b) { return { a[0] + b[0], a[1] + b[1] }; }
II operator-(II a, II b) { return { a[0] - b[0], a[1] - b[1] }; }
II operator*(II a, II b) { return { a[0] * b[0], a[1] * b[1] }; }
II operator%(II a, II b) { return { a[0] % b[0], a[1] % b[1] }; }

struct StrHash {
  int n;
  II base, mod;
  vector<II> pref, inv;

  StrHash(string s, II b = { 31, 37 }, \
  II m = { 1000000007, 1000000009 }): n(s.size()), \
  base(b), mod(m), pref(n+1), inv(n+1) {
    II sum = { 0, 0 };
    II pow = { 1, 1 };

    for (int i = 0; i < n; i++) {
      int c = s[i] - 'a' + 1;
      sum = (sum + II{ c, c } * pow % mod) % mod;
      pref[i+1] = sum;

      inv[i][0] = bin_exp(pow[0], mod[0]-2, mod[0]);
      inv[i][1] = bin_exp(pow[1], mod[1]-2, mod[1]);

      pow = pow * base % m;
    }
  }

  II substr_hash(int i, int j) { // [i, j)
    i = max(0ll, min(n, i));
    j = max(0ll, min(n, j));
    II hash = mod + pref[j] - pref[i];
    hash = hash % mod * inv[i] % mod;
    return hash;
  }
};
```

Min Cut

```
int n, m; c_in(n, m);
VII E(m); c_in(E);

VVI G(n+1);
auto M = VMAT(n+1, n+1);

for (auto [u, v]: E) {
  G[u].push_back(v);
  G[v].push_back(u);
  M[u][v] = 1;
  M[v][u] = 1;
```

```
}

auto max_flow = [&](int s, int t) {
  VC Vis(n+1);
  int delta = 1ll << 30;

  auto dfs = [&](int u, int fu, auto& self) -> int {
    if (u == t) return fu;
    Vis[u] = true;

    for (auto v : G[u]) if (!Vis[v] && M[u][v] >= delta) {
      int pushed = self(v, min(fu, M[u][v]), self);
      if (pushed) {
        M[u][v] -= pushed;
        M[v][u] += pushed;
        return pushed;
      }
    }
    return 0;
  };

  int flow = 0;

  while (delta) {
    int new_flow;
    do {
      fill(all(Vis), false);
      new_flow = dfs(s, INF, dfs);
      flow += new_flow;
    } while (new_flow);

    delta >>= 1;
  }

  return flow;
};

c_out(max_flow(1, n));

VII Cuts;
VC Vis(n+1);

auto dfs_cuts = [&](int u, auto& self) -> void {
  Vis[u] = true;
  for (auto v : G[u])
    if (!Vis[v] && M[u][v])
      self(v, self);
};

dfs_cuts(1, dfs_cuts);
```

Convex Hull (kaium)

```
II operator-(II p1, II p2) \
{ return { p1[0] - p2[0], p1[1] - p2[1] }; }

int orientation(II p1, II p2, II p) {
  II v1 = p2 - p1, v2 = p - p1;
  int cross = v1[0] * v2[1] - v1[1] * v2[0];
  return (cross > 0) - (cross < 0);
}

deque<II> convex_hull(vector<II>& V, \
bool collinear = false) {
  II lp = V[0]; // find the lowest point
  for (auto& p : V) if (p[1] < lp[1] || \
    (p[1] == lp[1] && p[0] < lp[0])) lp = p;

  deque<array<double, 3>> D;
  int n = V.size();
  // generate angle, distance, index for sorting
  FOR (i, 0, n-1) {
    II v = V[i] - lp;
    double angle = atan2(v[1], v[0]);
    double dist2 = v[0] * v[0] + v[1] * v[1];
    D.push_back({ angle, dist2, (double)i });
  }

  sort(all(D));

  // reverse collinear points at the back
  if (collinear) {
    double angle = D.back()[0];
    auto it = D.rbegin();
    while (it < D.rend() && (*it)[0] == angle) it++;
    reverse(D.rbegin(), it);
  }

  deque<II> H;
  int i; // add first two points
  i = D[0][2]; H.push_front(V[i]); D.pop_front();
  i = D[0][2]; H.push_front(V[i]); D.pop_front();

  for (auto& d : D) {
    i = d[2];
    II p = V[i];
    // < for keeping collinear points, <= otherwise
    while (orientation(H[1], H[0], p) < 0)
      H.pop_front();
    H.push_front(p);
  }

  reverse(all(H));
  return H;
}
```